

Примитивные типы

Конструкция ветвления

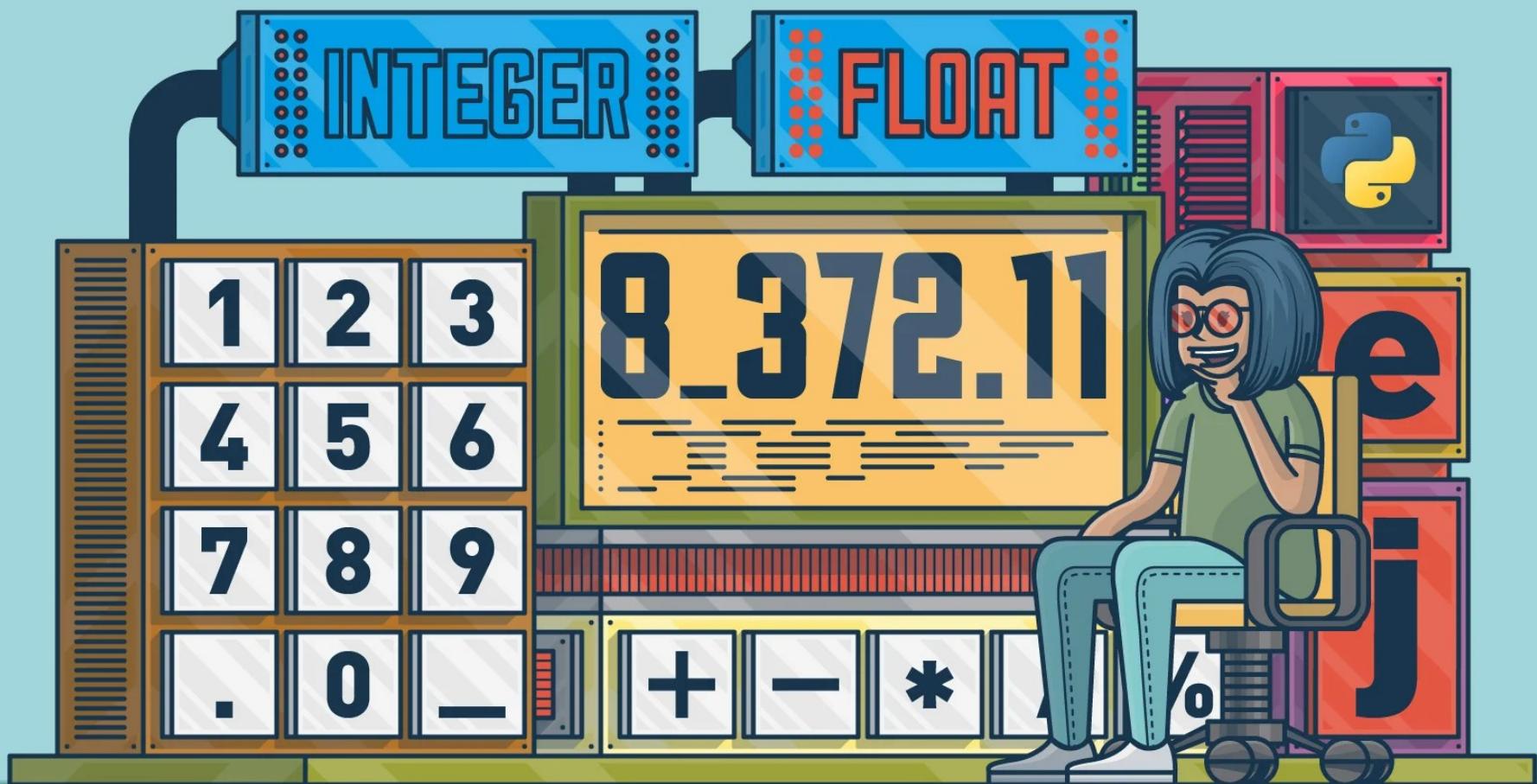
Оглавление

- Примитивные типы
- Таблица истинности
- Конструкции ветвления `if/elif/else`
- Тернарный оператор
- Оператор сопоставления с шаблонами `match/case`
(Python v. 3.10)

Примитивные типы

1. Целые числа (int)
2. Числа с плавающей запятой (float)
3. Комплексные числа (complex)
4. Строки (str)
5. Массивы байт (bytarray)
6. Логический (bool)
7. NoneType

NUMBER



Real Python

Операторы в Python для работы с числами

Операторы в Python для работы с числами:

"+"	(сложение)
"-"	(вычитание)
"*"	(умножение)
"/"	(деление)
"//"	(целочисленное деление)
"%"	(деление с остатком)
"**"	(возвведение в степень)

$\sqrt{25}$ эквивалентно `num ** (0.5)` -?

`abs(x)` - модуль числа

`divmod(x, y)` - пара $(x // y, x \% y)$

`pow(x, y[, z])` x^y по модулю (если модуль задан)

Системы счисления

- Десятичная

```
>>> 7          → int
```

```
>>> 3.14      → float
```

- Двоичная

```
>>> 0b0010    → int
```

- Восьмеричная

```
>>> 0o07      → int
```

- Шестнадцатиричная

```
>>> 0x0F      → int
```

Функции преобразования чисел

- `int(x)` - преобразование к целому числу в десятичной системе счисления.
- `bin(x)` - преобразование целого числа в двоичную строку.
- `hex(x)` - преобразование целого числа в шестнадцатеричную строку.
- `oct(x)` - преобразование целого числа в восьмеричную строку.

Особенности чисел

- Можно работать с большими числами

```
>>x=99999999999999999999999999999999999999999999999999999999999  
99999999999999999999999999999999999999999999999999999999999999999  
99999999999999999999999999999999999999999999999999999999999999999
```

- Форма записи числа

```
>> 9_999_999
```

- Если нужна + бесконечность (inf)

```
>> 2e400 → inf
```

- Если нужна - бесконечность (-inf)

```
>> -2e400 → -inf
```

Scientific notation

- Python использует нотацию Е для отображения больших чисел с плавающей запятой

```
>> 2000000000000000.0      →      2e+17
```

```
>> 1e15      → 1000000000000000.0
```

```
>> 1e16      → ?
```

```
>> 1e-4      → 0.0001
```

```
>> 1e-5      → ?
```

```
>> 1e-0      → ?
```

Юмор из Monty Python

```
>> 3.14 * 10           → 31.40000000000002
>> -4 // 3            → -2
>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
+ 0.1
0.999999999999999
```

Пример использования Decimal

```
>>> 0.1 + 0.1 + 0.1 - 0.3  
5.551115123125783e-17
```

Для высокой точности следует использовать другие объекты (например decimal и fraction)

```
>>> from decimal import Decimal  
>>> q=w=e=r=t=y=u=i=o=p=Decimal('0.1')  
>>> q+w+e+r+t+y+u+i+o+p  
Decimal('1.0')
```

Комплексные числа

- Комплексное число — это любое число в форме $a + bj$, где a и b — действительные числа, а $j*j = -1$.
- Каждое комплексное число ($a + bj$) имеет действительную часть (a) и мнимую часть (b).

```
>>n = 4 + 3j      → (4+3j)
```

Битовые операторы

~ битовый оператор НЕТ (инверсия, наивысший приоритет);

`<<, >>` – операторы сдвига влево или сдвига вправо на заданное количество бит;

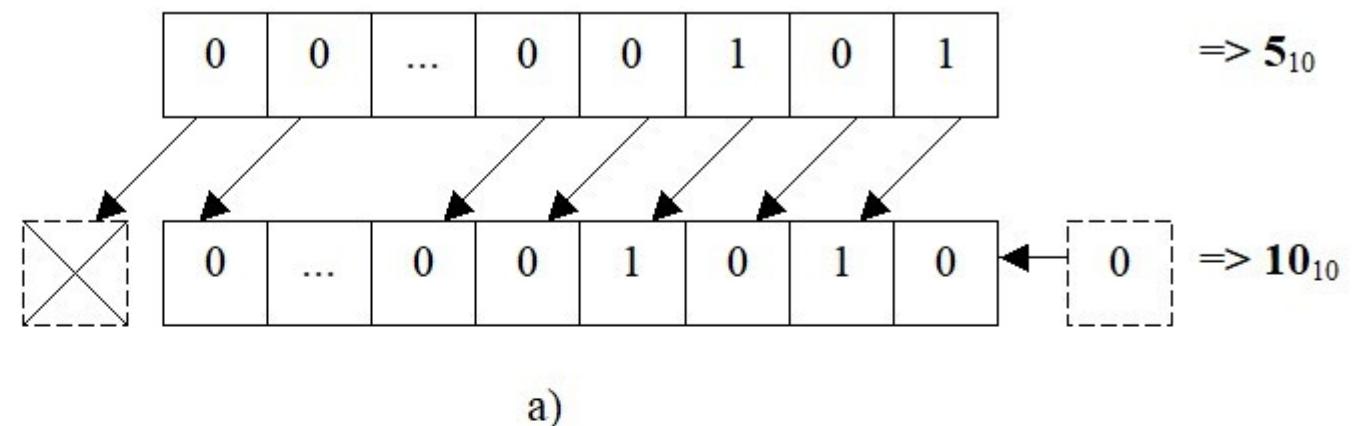
& битовый оператор И

^ битовое исключающее ИЛИ

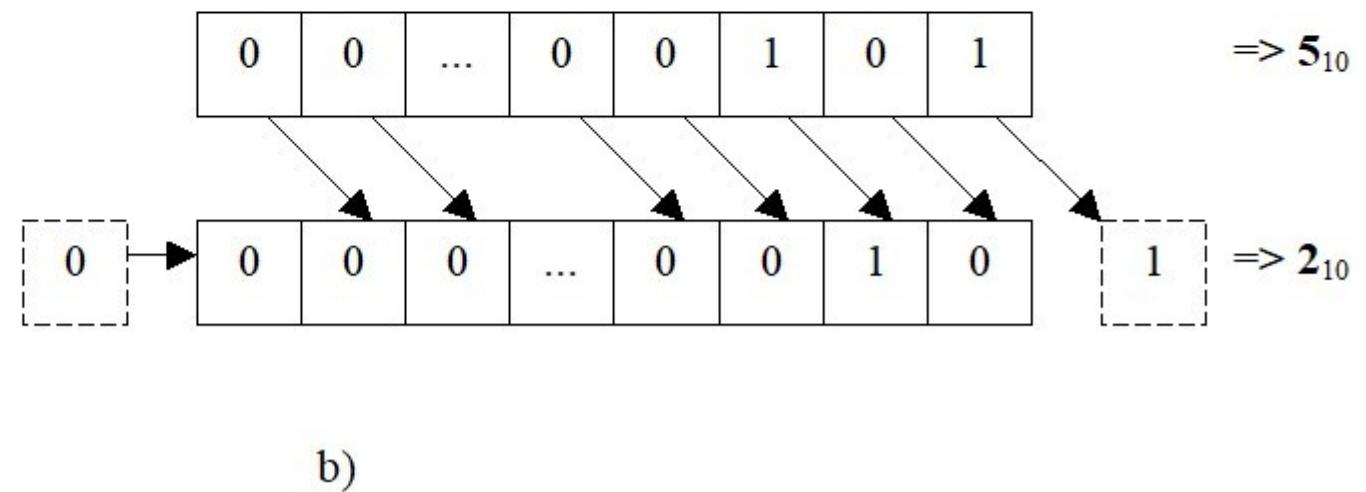
| битовый оператор ИЛИ.

Пример: Операторы сдвига влево `<<`, вправо `>>`

```
x = 5
y = 5 << 1 # y = 10
```



```
x = 5
y = x >> 1 # y = 2
```



Операторы сравнения

"=="	(равно)
">="	(больше или равно)
"<="	(меньше или равно)
"!="	(не равно)
"<"	(меньше)
">"	(больше)

Примечание: Когда мы хотим сравнить что две переменные равны то мы делаем так:

```
>> weight_one = 100
>> weight_two = 100
>> weight_one == weight_two      →      true
>> weight_one != 90            →      true
>> weight_one = weight_two      ←      не правильно !!
```

Строки- последовательности символов

Unicode

Строка задается либо парой одинарных " " , либо двойных " " " " или тройных " " " " " " ковычек. Существенный разницы в Python между одинарными и двойными ковычками нет.

```
>>> name = "" → пустая строка
>>> name = """"" → пустая строка
>>> print(""""
Usage: thingy [OPTIONS]
      -h      Display this usage message
      -H hostname      Hostname to connect to
""")
>>> name = "Peter I"
```

Внимание !!! Частая ошибка.

Не забывайте ковычки при задании строки,
иначе значение будет интерпретироваться
как переменная.

```
>>> str = Hello
```



Максимальная длина строки в Python

Максимальная длина строки зависит от платформы. Обычно это:

- $2^{31} - 1$ — для 32-битной платформы;
- $2^{63} - 1$ — для 64-битной платформы;

Константа `maxsize`, определенная в модуле `sys` :

```
>>> import sys >>> sys.maxsize 2_147_483_647
```

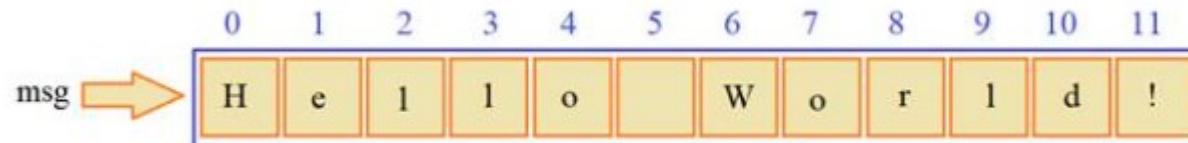
ФУНКЦИЯ `len()` вычисляет длину строки.

```
>>> len("Hello")      → 5
```

Индексация строки

- Для получения символа в строке нужно обратиться по индексу позиции. Индексация строк начинается с 0

```
>>msg           = "Hello World!"  
>>msg[0]        → "H"
```



Операции сложения и умножения строк

Строки можно складывать (конкатенация строк)

```
>>string = "Hello" + " world !" → "Hello world!"
```

Строки можно умножать на целые числа. Происходит повторение строки n раз.

```
>>> "NO!" * 3 → 'NO!NO!NO!'
```

IMMUTABLE

Строка является неизменяемой (immutable) последовательностью символов.

```
>>msg = "Hello World!"
```

Попытка записать значение в начало слова, вызовет ошибку!

```
>>msg[0] = "S"
```

```
TypeError: 'str' object does not support item assignment
```

Строки (str). Срезы (slices)

Срезы (slices) – извлечение из данной строки одного символа или некоторого фрагмента (подстроки)

0	1	2	3	4
H	E	L	L	O
-5	-4	-3	-2	-1

Оператор извлечения среза из строки выглядит так: [X:Y].
X – индекс **начала среза**,
Y – индекс **окончания среза**(символ с номером Y в срезе не входит).

```
>>> s = 'hello'          >>> s = 'hello'  
>>> s[1:4]      ИЛИ      >>> s[-4:-1]  
'ell'           'ell'  
>>>                  >>>
```

Срезы (slice)

#Пустое значение в начале обозначает позицию 0 индекса

```
>>>str = "Hello !"  
>>>str[:3]  # Соберем срез по индексам 0, 1, 2  
"He1"  
>>>str[None:3]  # Аналогично
```

#Пустое значение в конце обозначает позицию по концу строки

```
>>str[3:]  # Соберем срез по индексам от 2 до 6  
>>"lo !"
```

Проверка вхождения значения в последовательность.

```
>>> "P" in "Python"  
True
```

```
>>> "world" in "Hello world"  
True
```

Сравнение строк при помощи == и !=

```
>>>language = 'chinese'  
>>>print(language == 'chinese')      → True  
>>>print(language != 'chinese')     → False
```

```
>>> 'chinese' > 'italiano'
```

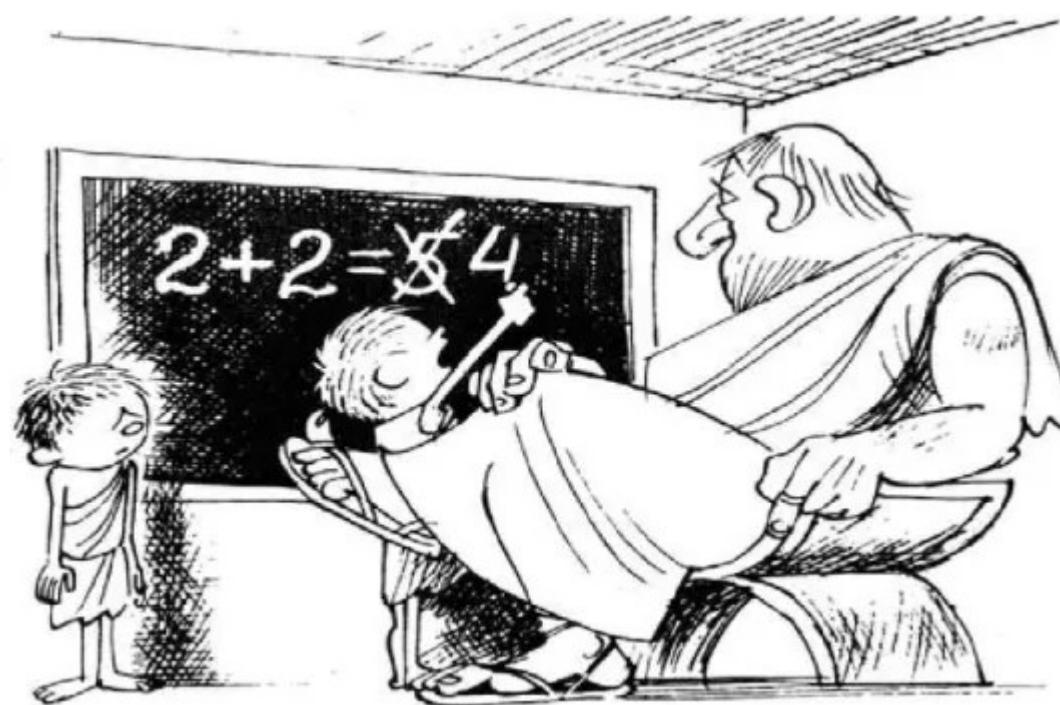
Ответ: ?

Массивы байт.

Bytarray в python – массив байт. От строк отличается только тем, что является изменяемым.

```
>>> b = bytarray(b'hello world!')  
>>> b  
bytarray(b'hello world!')  
>>> b[0]  
104  
>>> b[0] = b'h'  
Traceback (most recent call last):  
  File "", line 1, in  
    b[0] = b'h'  
TypeError: an integer is required  
>>> b[0] = 105  
>>> b  
bytarray(b'iello world!')
```

Питон мне друг но истина дороже.



Логические операторы

AND

— логическое И

OR

— логическое ИЛИ

NOT

— логическое отрицание

IN

— возвращает истину, если элемент присутствует в последовательности, иначе ложь.

NOT IN

— возвращает истину если элемента нет в последовательности.

IS

— проверка идентичности объекта

Таблица истинности

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Python - Logical Operators

- not

x	not x
False	True
True	False

- and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

- or

x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True



<http://inderpsingh.blogspot.com/>

Применение логических операторов

```
x = 10  
y = 20
```

```
if x > 0 and y > 0:  
    print('Положительные числа')
```

```
if x > 0 or y > 0:  
    print('Хотя бы одно положительное')
```

```
if x > (0 or y) > 0:  
    print('Что будет')
```

Таблица приоритетов операций

Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+X, -X, ~X	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
	l	Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, Identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

Вывод

Чтобы не запутаться в приоритетах операций ставьте в выражении круглые скобки ()

```
# Тестирование порядка выполнения выражения ( слева направо)
```

```
print(4 * 7 % 3)
```

```
# Результат: 1
```

```
print(2 * (10 % 5))
```

```
# Результат: 0
```



Конструкции ветвления if/else

```
age = input("Enter your age")
age = int(age)
if age <= 18:
    print("Доступ запрещен!")
else:
    print("Доступ разрешен!")
```

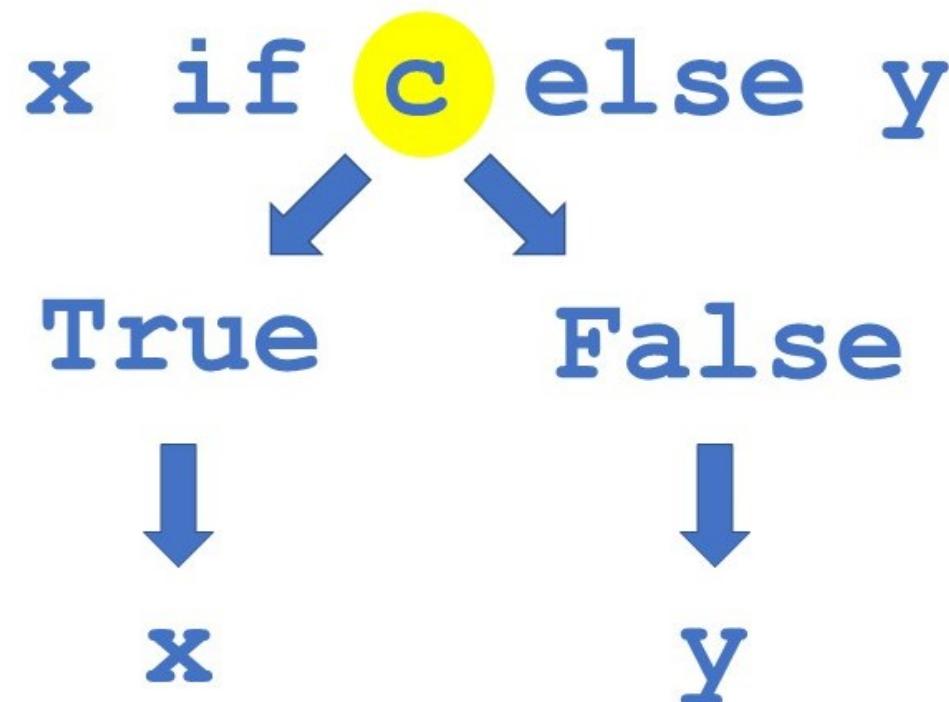
Конструкции ветвления if/elif/else

```
age = input("Enter your age")
age = int(age)
if age <= 16:
    print("Школьник!")
elif 16 < age <= 25:
    print("Студент")
elif 25 < age <= 40:
    print("Сотрудник компании")
else:
    print("Еще не существует!")
```

Конструкции ветвления if

```
str = "Hello"  
if str:  
    print("Не пустая строка!")
```

Ternary Operator



Тернарный оператор

```
x = 1
```

```
y = 2
```

```
maximum = x if x > y else y
```

Pattern Matching in Python 3.10



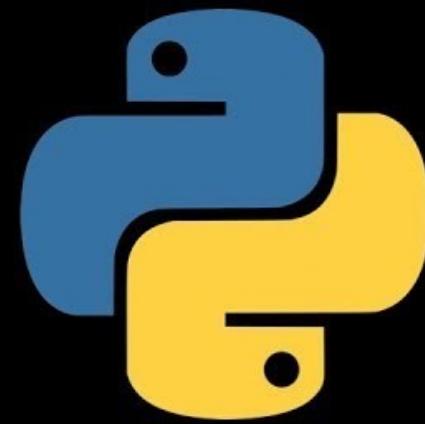
PEP 622

```
match status:  
    case 200:  
        print("OK")  
    case 301 | 302:  
        print("Redirect")  
    case 404:  
        print("Not Found")
```

Пример:

```
color = "RED"

match color:
    case "RED":
        print("Флаг красный")
    case "GREEN":
        print("Трава зеленая")
    case "BLUE":
        print("Небо синее")
```



PYTHON PROGRAMMING

Задача 1

Дано целое число. Если оно является положительным, то прибавить к нему 1; если отрицательным, то вычесть из него 2; если нулевым, то заменить его на 10. Вывести полученное число.

Задача 2.

Задано целое число от 10 до 99 ,
нобходимо найти сумму и произведение
его цифр

Пример:

$$x = 23 \rightarrow 2 + 3$$

Ответ:

сумма - 5

произведение - 6

Задача 3

Задана произвольная строка.
Необходимо разбить ее на две части .
Задачу решить с помощью срезов.

Пример:

str = “Hello world!”

Ответ: Первая часть “Hello”
Вторая часть “world!”